ReVive:
Cost-Effective Architectural Support for Rollback Recovery in Shared-Memory Multiprocessors

Milos Prvulovic, Zheng Zhang*, Josep Torrellas

University of Illinois at Urbana-Champaign
*Hewlett-Packard Laboratories

Motivation

- Availability & Reliability increasingly important
- Frequency $\uparrow$, Feature Size $\downarrow$ $\Rightarrow$ Errors $\uparrow$
- Complexity $\uparrow$, Verification Cost $\uparrow$ $\Rightarrow$ Errors $\uparrow$
- Multiprocessors $\Rightarrow$ Errors $\uparrow$
- Global software-only recovery too slow
- Can hardware help?

Motivation

- Cost vs. Performance vs. Availability
- Low Cost
  – Simple changes to a few key components
- Low Performance Overhead
  – Handle frequent operations in hardware
- High Availability
  – Fast recovery from a wide class of errors

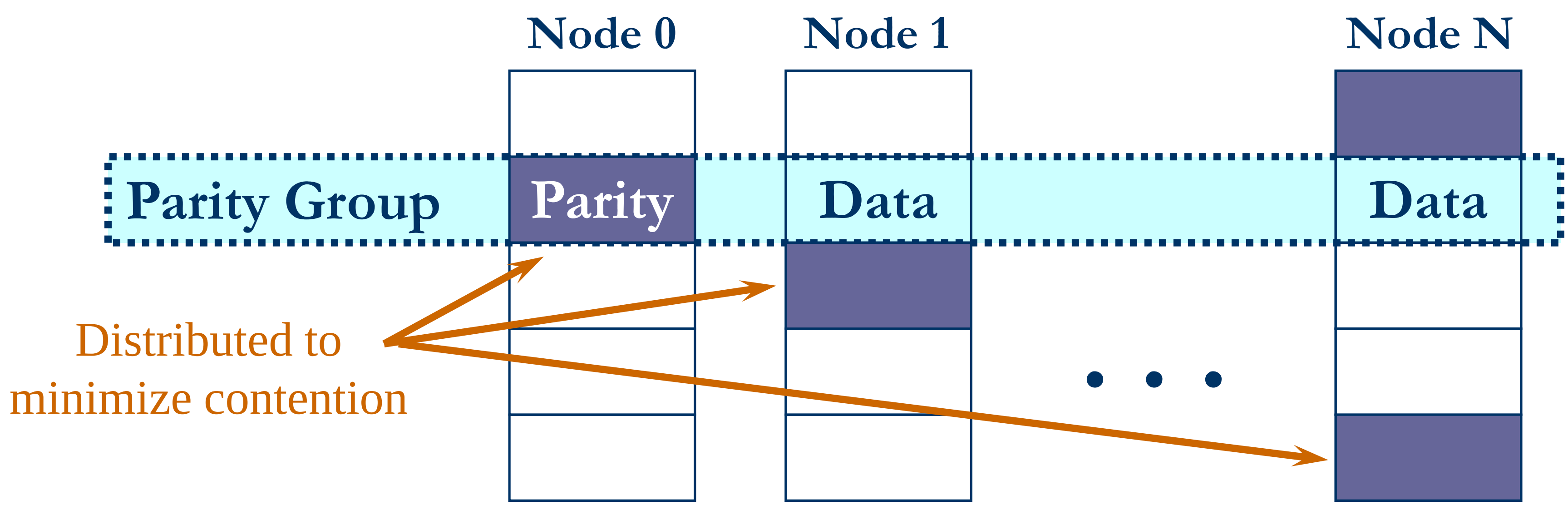
Contribution: New Scheme

- Low Cost
  - HW changes only to directory controllers
  - Memory overhead only 12.5% (with 7+1 parity)
- Low Performance Overhead
  - Only 6% performance overhead on average
- High Availability
  - Recovery from: system-wide transients, loss of one node
  - Availability better than 99.999% (assuming 1 error/ day)

Overview of ReVive

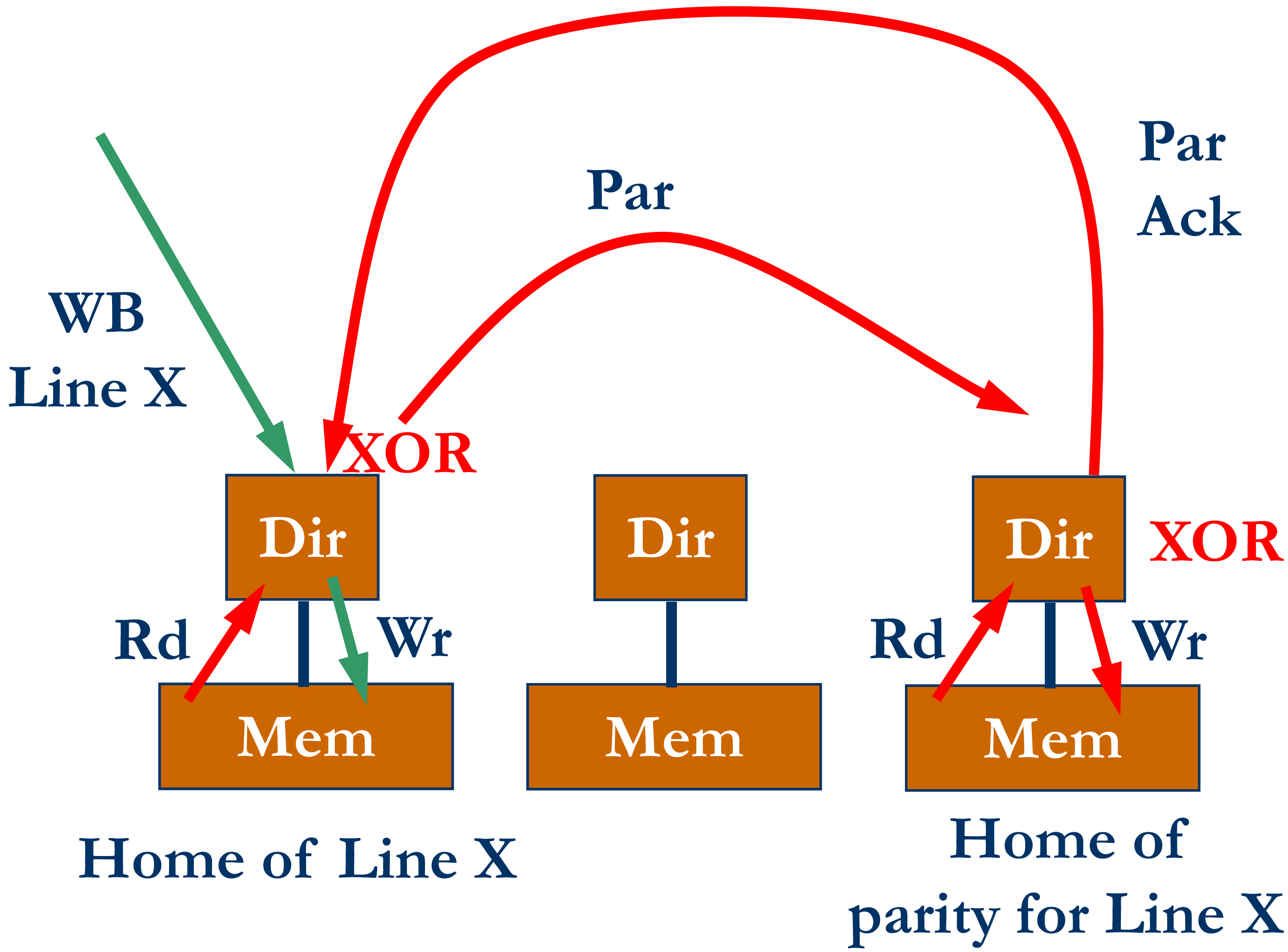
- Entire main memory protected by distributed parity
  - Like RAID-5, but in memory
- Periodically establish a checkpoint
  - Main memory is the checkpoint state
  - Write-back dirty data from caches, save processor context
- Save overwritten data to enable restoring checkpoint
  - When program execution modifies memory for 1st time

Distributed N+1 Parity

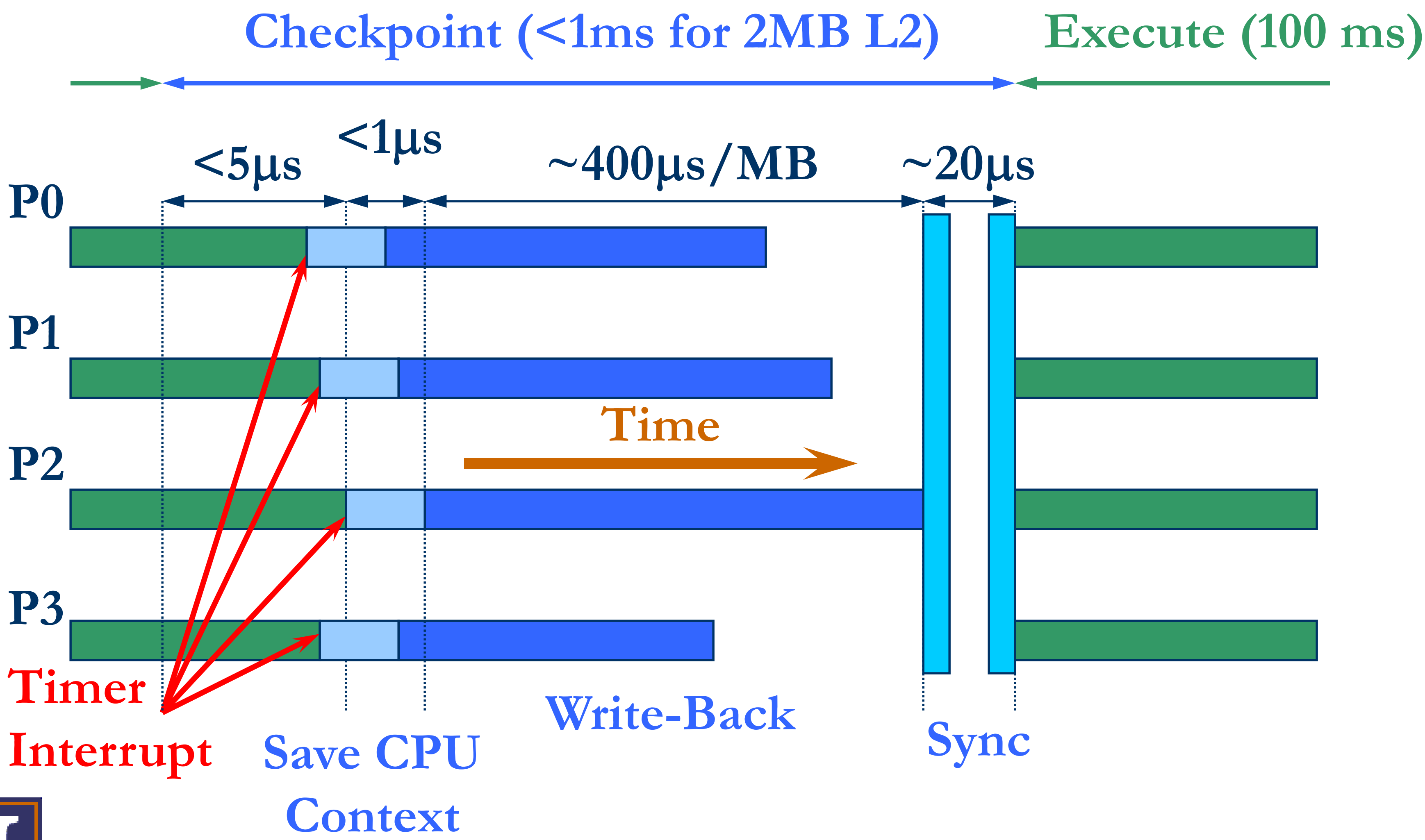


- Allocation Granularity: page
- Update Granularity: cache line

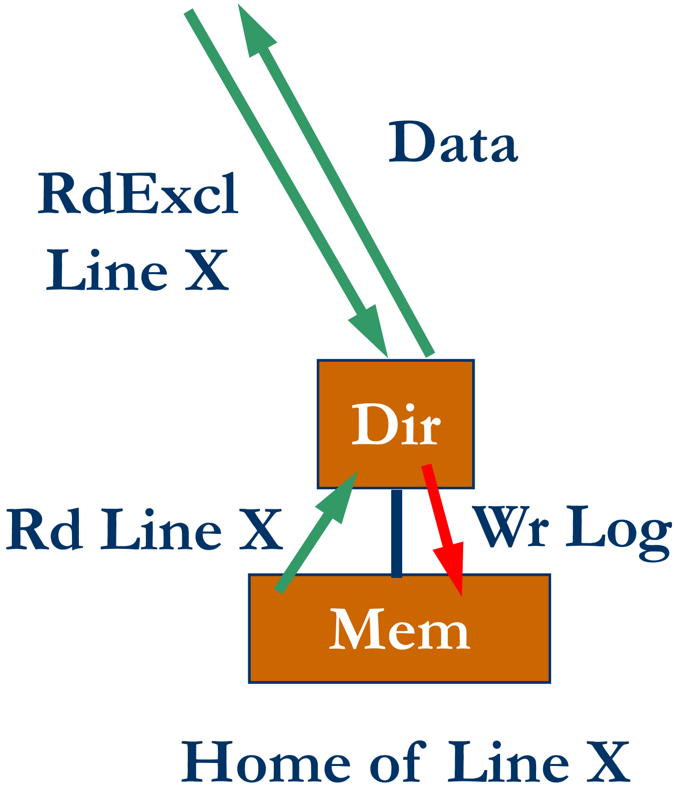
Distributed Parity Update in HW



ReVive: Checkpoint Creation Timeline



Logging in HW

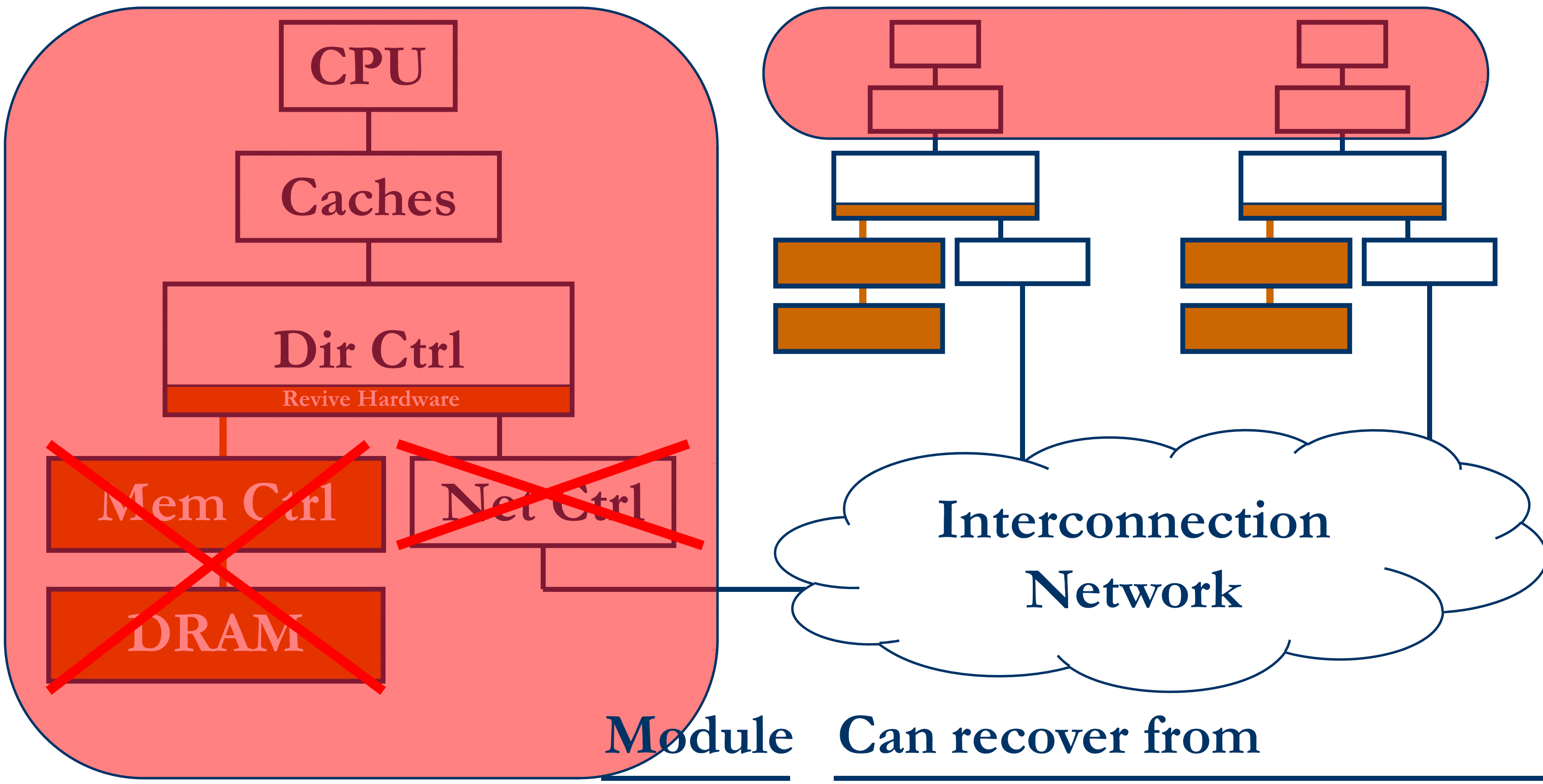


Note:
Wr Log also updates the parity

Log Filtering

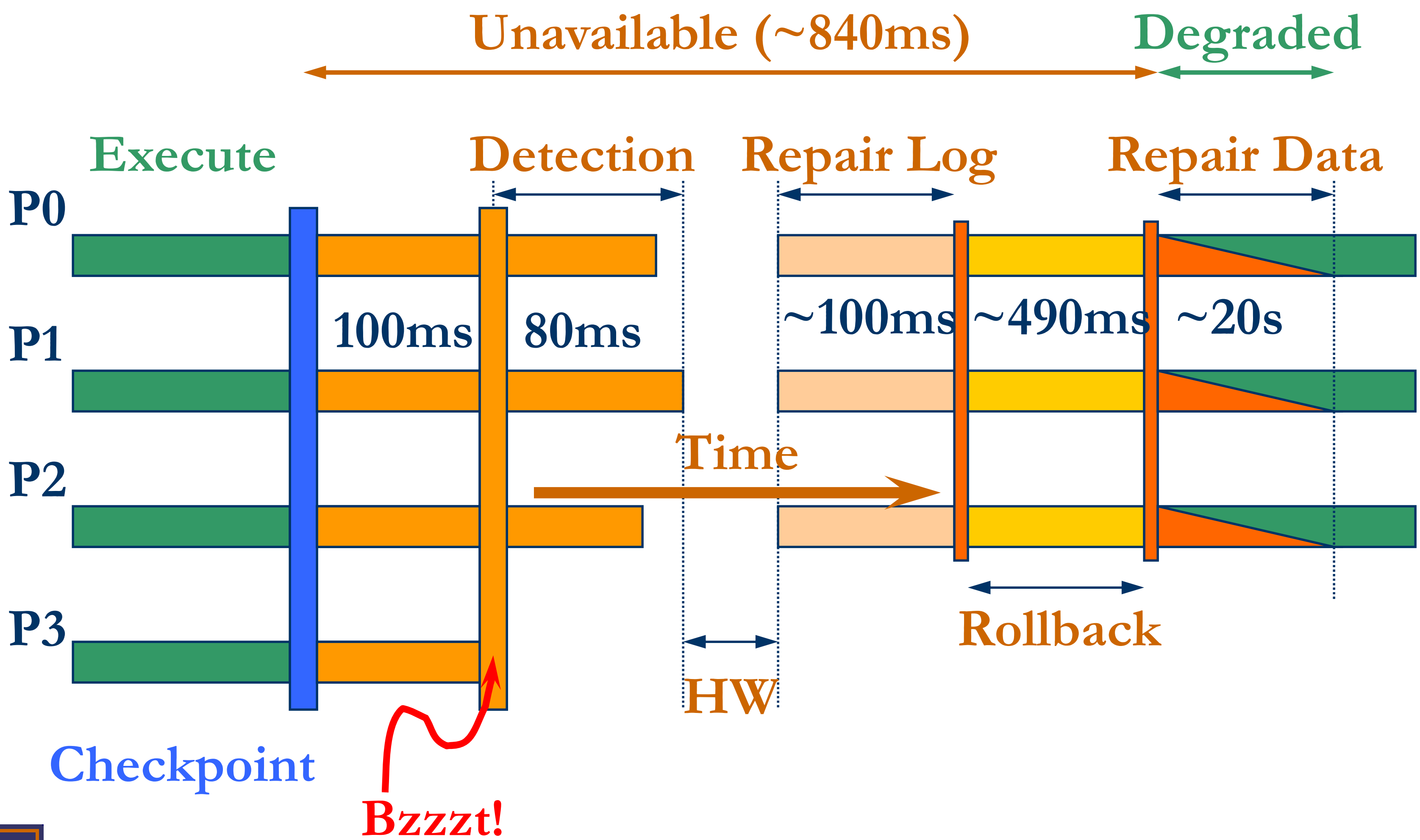
- Add L bit to directory entry of each line
  - Clear all L bits on each checkpoint
  - Set when logged
  - Do not log if already set
- Not needed for correctness
  - Can be only in directory cache
  - Can be completely omitted

Classes of Recoverable Errors



Module	Can recover from
■ □	(Trans + perm) errors in 1 node
□	Trans errors in N nodes

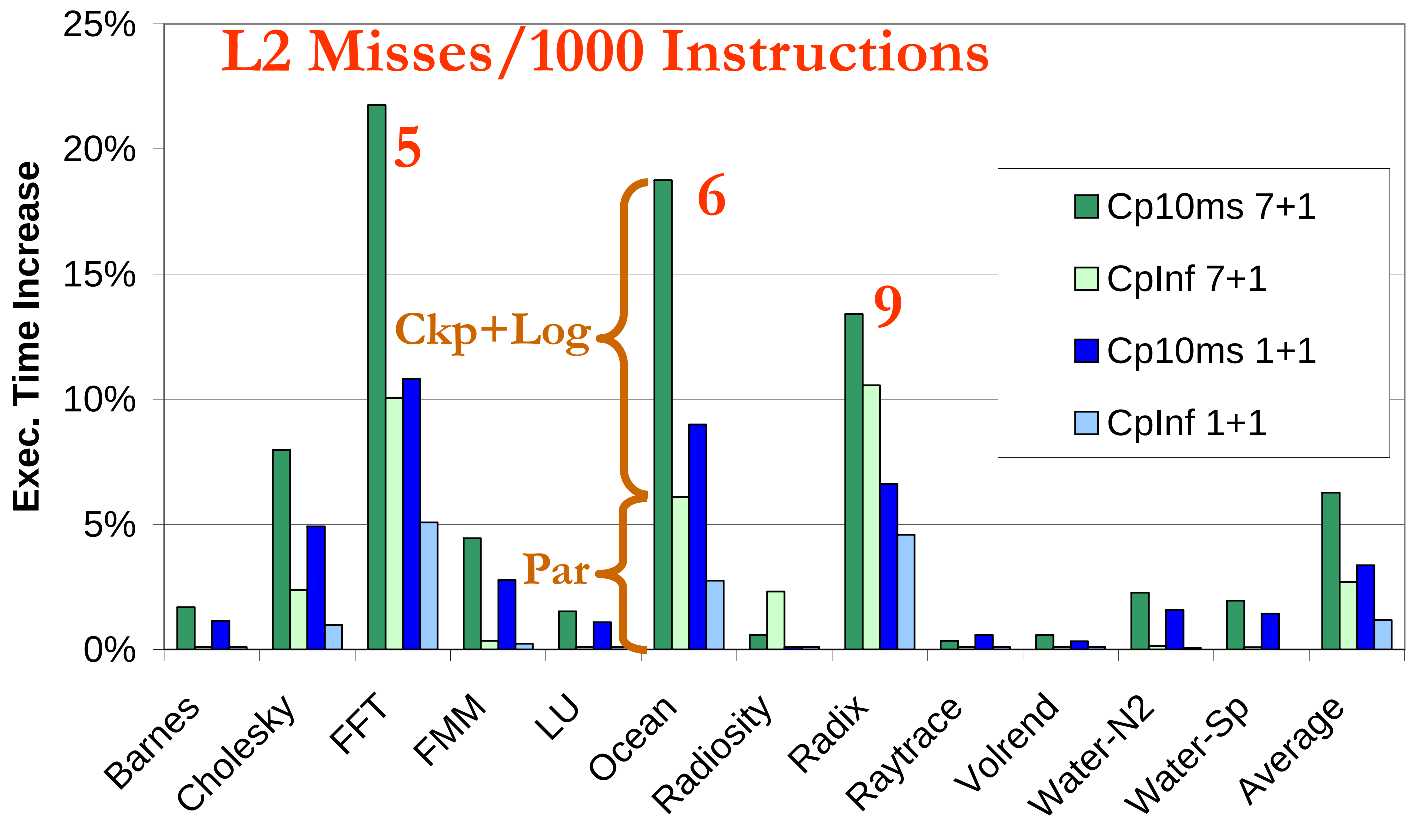
Permanent Node Loss: Recovery



Evaluation Setup

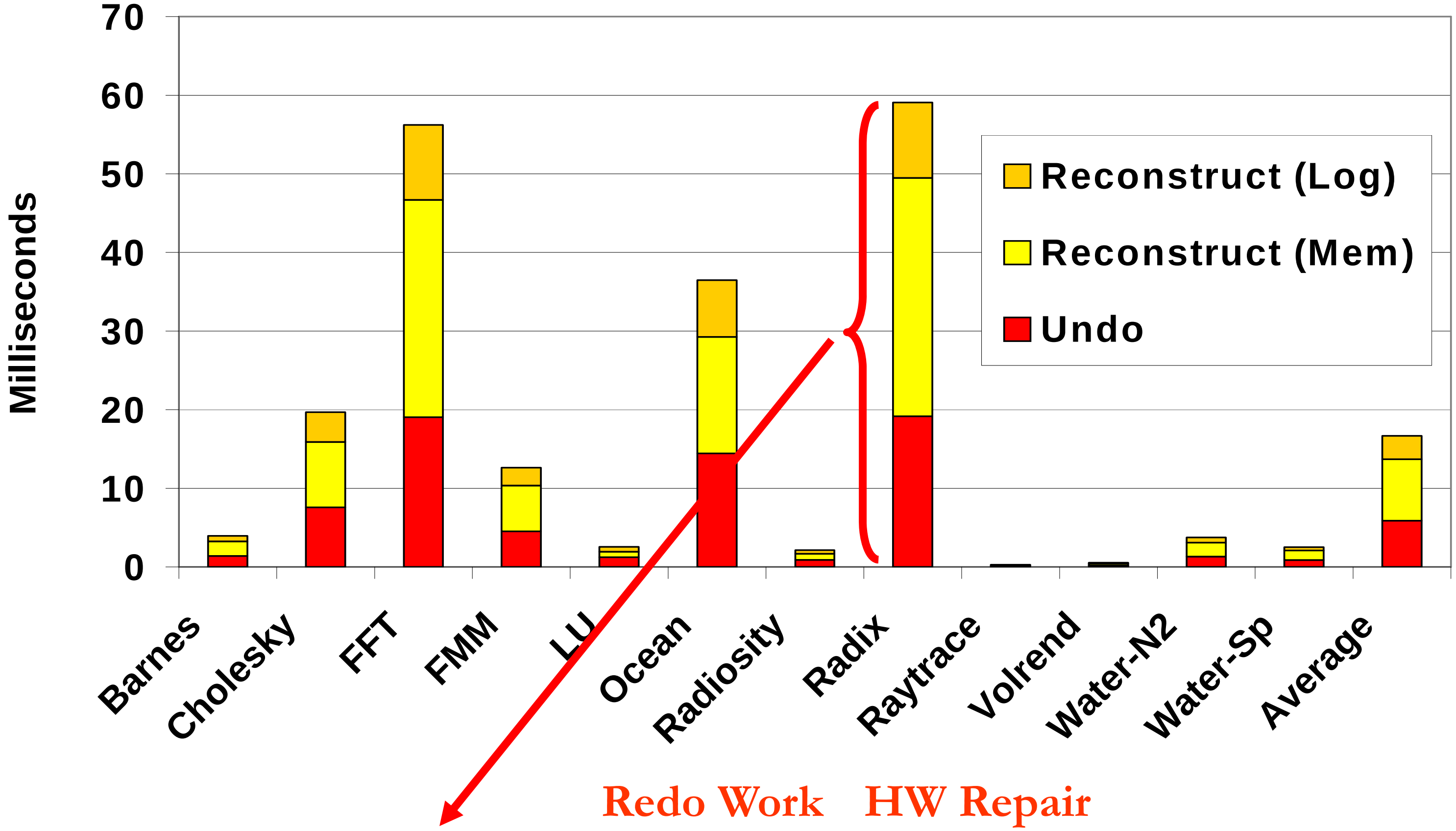
- Splash-2 benchmarks
- 16 superscalar processors (6-issue at 1GHz)
- 16kB L1 cache, 512kB L2 cache
- 2-D torus network, virtual cut-through routing
- 100MHz DDR SDRAM
- Using 7+1 distributed parity
- Checkpoint interval: 10ms and infinite

Performance Overhead



Tolerable 6% performance overhead

Worst-Case Recovery Time

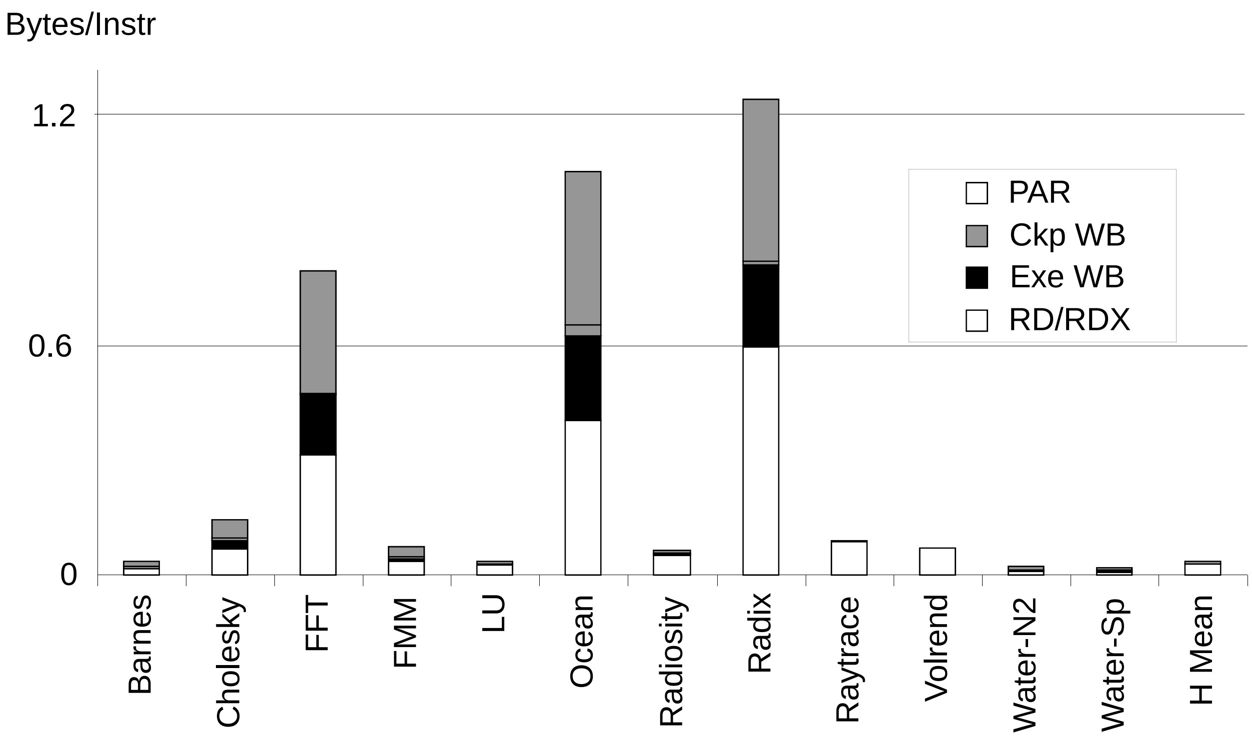


Redo Work HW Repair

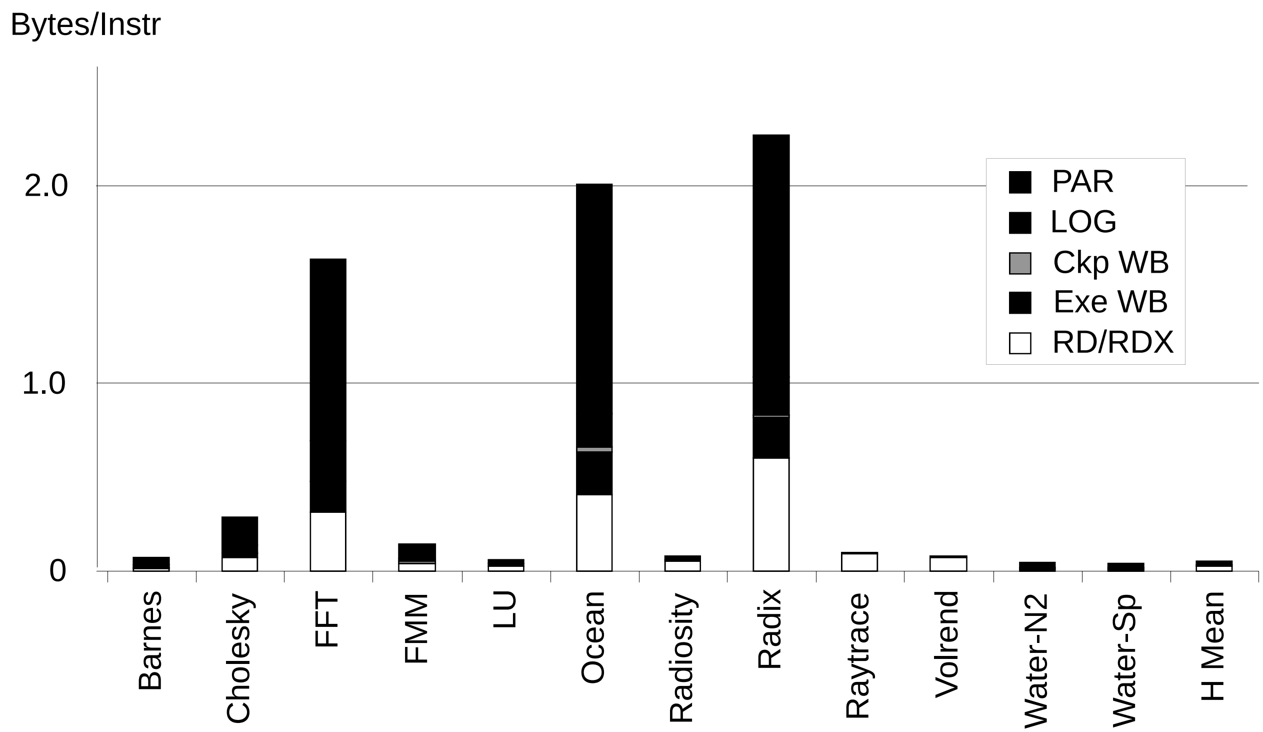
- Radix: 590ms + 180ms + 50ms = 820ms

⇒ 99.999% availability

Network Traffic



Memory Traffic



Related Work

- Device- or problem-specific schemes
  - DIVA, Redundant Multithreading, Slipstream, ECC, etc.
  - ReVive can handle errors that escape these schemes, improving overall availability at low additional cost
- Other system-recovery schemes
  - Plank et al. - N+1 parity in software
  - Masubuchi et al. - logging with bus-snooper
  - SafetyNet

Related Work: SafetyNet

- Types of recoverable errors
  - **ReVive**: Permanent (loss of a node)+Transient
  - **SafetyNet**: Transient; perm only w/ redundant devices
- HW modifications
  - **ReVive**: Directory controller only
  - **SafetyNet**: Memory, caches, coherence protocol
- Performance Overhead
  - 6% with **ReVive**, negligible with **SafetyNet**

Conclusions

- Recovery from: system-wide transients, loss of 1 node
- Availability better than 99.999%
- Low performance overhead (6% on average)
- HW changes only to directory controllers
- Memory overhead 12.5% with 7+1 parity
  - Overhead can be reduced by increasing parity groups

ReVive:
Cost-Effective Architectural Support for Rollback Recovery in Shared-Memory Multiprocessors

Milos Prvulovic, Zheng Zhang, Josep Torrellas

http://iacoma.cs.uiuc.edu
prvulovi@cs.uiuc.edu

Rollback Recovery in Multiprocessors

- Checkpoint Consistency
  - Global, Local Coordinated or Local Uncoordinated
- Checkpoint Separation
  - Full or Partial
  - Partial can be with Logging, Renaming or Buffering
- Checkpoint Storage
  - Safe External, Safe Internal or for a Specialized Error Class

Checkpoint Consistency

- Global **Synchronization is fast enough on shared-memory machines**
  - All synchronize to make a single consistent checkpoint
- Local Coordinated
  - Synchronize as needed for a set of consistent checkpoints
- Local Uncoordinated
  - Do not synchronize
  - Set of consistent checkpoints computed when recovering

Checkpoint Storage

- Safe External (e.g. RAID) **Not fast enough**
  - Recovery data on redundancy protected-disk
- Safe Internal (e.g. DRAM)
  - Recovery data in redundancy-protected memory
- Unsafe Internal **Not general enough**
  - Recovery data not protected by redundancy
  - Assumes memory content survives errors

Checkpoint Separation

- Full **Too much storage needed**
  - Checkpoint and working data sets do not intersect
- Partial with Buffering **Commit atomicity, overhead**
  - Buffer non-checkpoint data, flush to commit
- Partial with Renaming **Complex HW or coarse grain**
  - Rename to avoid overwriting checkpoint data
- Partial with Logging
  - Save overwritten checkpoint data in a log

Log & Parity Update Races

- Error while log update in progress
  - Must fully perform log update before starting overwrite
- Error while parity update in progress
  - Assume a single node fails
  - Can recover either old or new content
  - Both result in consistent recovery (see paper)
- Long error detection latency
  - Keep sufficient logs to recover far enough into the past

Availability vs Overhead

- If checkpoint interval too short
  – Lost work and hardware self-check dominate recovery
  – Fault-free execution performance suffers
- If checkpoint interval too long
  – Low availability
- Find a good balance
  – Checkpoint intervals of 100ms to 1s

Analysis

- Cache size vs. checkpoint interval
  - 512kB caches with checkpoints every 10ms
  - 5MB caches with checkpoints every 100ms
- Log size vs. checkpoint interval
  - Log will grow in sub-linear proportion to interval size
  - 10ms: <3MB per node, only two apps >128kB per node
- Parity overhead: 12.5% of system memory is parity